

[illegible]

```
MM      MM      FFFFFFFF      SSSSSSSS      UU      UU      MM      MM      MM      MM
MM      MM      FFFFFFFF      SSSSSSSS      UU      UU      MM      MM      MM      MM
MMMM    MMMM    FF      SS      UU      UU      MMMM    MMMM    MMMM    MMMM
MMMM    MMMM    FF      SS      UU      UU      MMMM    MMMM    MMMM    MMMM
MM      MM      FF      SS      UU      UU      MM      MM      MM      MM
MM      MM      FF      SS      UU      UU      MM      MM      MM      MM
MM      MM      FFFFFFFF      SSSSSS      UU      UU      MM      MM      MM      MM
MM      MM      FFFFFFFF      SSSSSS      UU      UU      MM      MM      MM      MM
MM      MM      FF      SS      UU      UU      MM      MM      MM      MM
MM      MM      FF      SS      UU      UU      MM      MM      MM      MM
MM      MM      FF      SS      UU      UU      MM      MM      MM      MM
MM      MM      FF      SS      UU      UU      MM      MM      MM      MM
MM      MM      FF      SSSSSSSS      UU      UU      MM      MM      MM      MM
MM      MM      FF      SSSSSSSS      UUUUUUUUUU      MM      MM      MM      MM
MM      MM      FF      SSSSSSSS      UUUUUUUUUU      MM      MM      MM      MM
```

```
....
....
....
....
```

```
LL      IIIIII      SSSSSSSS
LL      IIIIII      SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLLLL      IIIIII      SSSSSSSS
LLLLLLLLLL      IIIIII      SSSSSSSS
```

```
1 MFSUM_REQUEST: Procedure Returns(Fixed Binary(31)) /* Routine to execute multi-file summary req
2 Options(Ident('V04-000'));
```

```
3
4 /*
5 *****
6 /*
7 /* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
8 /* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
9 /* ALL RIGHTS RESERVED.
10 /*
11 /* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
12 /* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
13 /* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
14 /* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
15 /* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
16 /* TRANSFERRED.
17 /*
18 /* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
19 /* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
20 /* CORPORATION.
21 /*
22 /* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
23 /* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
24 /*
25 /*
26 *****
27 */
28
29 /*
30 /*+++
31 /* FACILITY: MONITOR Utility
32 /*
33 /* ABSTRACT: MFSUM_REQUEST Routine.
34 /*
35 /*          Called from MONMAIN routine to execute a single
36 /*          MONITOR multi-file summary request.
37 /*
38 /*
39 /* ENVIRONMENT:
40 /*
41 /*          Unprivileged user mode.
42 /*
43 /* AUTHOR: Thomas L. Cafarella, January, 1984
44 /*
45 /*
46 /* MODIFIED BY:
47 /*
48 /*          V03-006 TLC1088      Thomas L. Cafarella      25-Jul-1984      14:00
49 /*          Free virtual memory obtained for multi-file summary.
50 /*
51 /*          V03-005 TLC1077      Thomas L. Cafarella      11-Jul-1984      14:00
52 /*          Continue multi-file summary even if some input files are empty.
53 /*
54 /*          V03-004 TLC1071      Thomas L. Cafarella      17-Apr-1984      14:00
55 /*          Add "number of input files" to multi-file summary report.
```


MFSUM REQUEST
V04-000

K 3
16-SEP-1984 02:18:09
5-SEP-1984 15:09:17

VAX-11 PL/I X2.1-273
DISK\$VMSMASTER:[MONITOR.SRC]MFSUMM.PLI;1 (1) Page 2

56	:	1	/*				
57	:	1	/*	V03-003	TLC1069	Thomas L. Cafarella	13-Apr-1984 14:00
58	:	1	/*		Fix ACCVIO when multi-file summary requested for SYSTEM class		
59	:	1	/*		and single-statistic.		
60	:	1	/*				
61	:	1	/*	V03-002	PRS1021	Paul R. Senn	12-Apr-1984 10:00
62	:	1	/*		Make SYSTEM class work with multi-file summary.		
63	:	1	/*				
64	:	1	/*	V03-002	PRS1020	Paul R. Senn	12-Apr-1984 10:00
65	:	1	/*		Make XQP class work with multi-file summary.		
66	:	1	/*				
67	:	1	/*	V03-002	TLC1060	Thomas L. Cafarella	12-Mar-1984 11:00
68	:	1	/*		Make multi-file summary work for homogeneous classes.		
69	:	1	/*				
70	:	1	/*	V03-001	TLC1053	Thomas L. Cafarella	07-Mar-1984 11:00
71	:	1	/*		Add support to ignore unused files for multi-file summary.		
72	:	1	/*				
73	:	1	/*--				
74	:	1	/*				
75	:	1					

MFSUM_REQUEST
V04-000

L 3
16-SEP-1984 02:18:09
5-SEP-1984 15:09:17

VAX-11 PL/I X2.1-273 Page 3
DISK\$VMSMASTER:[MONITOR.SRC]MFSUMM.PLI;1 (2)

```
76 | 1 | /*++  
77 | 1 | /*  
78 | 1 | /* FUNCTIONAL DESCRIPTION:  
79 | 1 | /*  
80 | 1 | /*     MFSUM_REQUEST  
81 | 1 | /*  
82 | 1 | /*     TBS  
83 | 1 | /*  
84 | 1 | /* INPUTS:  
85 | 1 | /*  
86 | 1 | /*     None  
87 | 1 | /*  
88 | 1 | /* IMPLICIT INPUTS:  
89 | 1 | /*  
90 | 1 | /*     Monitor Request Block (MRB), pointed to by MRBPTR.  
91 | 1 | /*  
92 | 1 | /* OUTPUTS:  
93 | 1 | /*  
94 | 1 | /*     None  
95 | 1 | /*  
96 | 1 | /* IMPLICIT OUTPUTS:  
97 | 1 | /*  
98 | 1 | /*     TBS  
99 | 1 | /*  
100 | 1 | /* ROUTINE VALUE:  
101 | 1 | /*  
102 | 1 | /*     SSS_NORMAL, or failing MONITOR status code.  
103 | 1 | /*  
104 | 1 | /* SIDE EFFECTS:  
105 | 1 | /*  
106 | 1 | /*     None  
107 | 1 | /*  
108 | 1 | /*/  
109 | 1 |
```

```
110 1 /*
111 1 /*
112 1 /*
113 1 /*
114 1 /*
115 1 /*
116 1 /*/
117 1
118 1 %INCLUDE MONDEF; /* Monitor utility structure definitions */
886 1 %INCLUDE $CHFDEF; /* Condition handler facility definitions */
906 1 %INCLUDE $STSDEF; /* Status value definitions */
923 1 %INCLUDE SYSS$CANCEL; /* $CANCEL system service */
929 1 %INCLUDE SYSS$PUTMSG; /* $PUTMSG system service */
937 1
938 1
939 1 /*
940 1 /*
941 1 /*
942 1 /*
943 1 /*
944 1 /*
945 1 /*/
946 1
947 1 Declare
948 1 MAX_CLASS_NO FIXED BINARY(31) GLOBALREF VALUE; /* Maximum defined class number */
949 1 PROC$CLSNO FIXED BINARY(31) GLOBALREF VALUE; /* Class number for the PROCESSES class */
950 1 MAXELTS_MFS FIXED BINARY(31) GLOBALREF VALUE; /* Max no. of elements for a homogeneous class m.f.
951 1 MAX_ELIDLEN FIXED BINARY(31) GLOBALREF VALUE; /* Max length of an element id for a homogeneous cla
952 1 MAX_HOM_ITEMS FIXED BINARY(31) GLOBALREF VALUE; /* Max no. of items for a homogeneous class */
953 1 MAX_INP_FILES FIXED BINARY(31) GLOBALREF VALUE; /* Max no. of input files for multi-file summary */
954 1 ECOUNT_SYS_ALL FIXED BINARY(31) GLOBALREF VALUE; /* Element count for SYS/ALL */
955 1
956 1 Declare
957 1 CDBPTR POINTER GLOBALREF; /* Pointer to CDB (Class Descriptor Block) */
958 1 C POINTER DEFINED(CDBPTR); /* Synonym for CDBPTR */
959 1 MCAPTR POINTER GLOBALREF; /* Pointer to MCA (Monitor Communication Area) */
960 1 MC POINTER DEFINED(MCAPTR); /* Synonym for MCAPTR */
961 1 MRBPTR POINTER GLOBALREF; /* Pointer to MRB (Monitor Request Block) */
962 1 M POINTER DEFINED(MRBPTR); /* Synonym for MRBPTR */
963 1
964 1 Declare
965 1 H POINTER GLOBALREF; /* Pointer to input file header */
966 1
967 1 Declare
968 1 INPUT_FILE FILE RECORD INPUT; /* Monitor Input (Playback) File */
969 1
970 1 Declare
971 1 CTRLCZ_CHAN FIXED BINARY(31) GLOBALREF; /* Channel number for CTRL-C and CTRL-Z */
972 1 CTRLCZ_HIT BIT(1) ALIGNED GLOBALREF; /* YES => CTRL-C or CTRL-Z has been hit */
973 1 NORMAL FIXED BINARY(31) GLOBALREF; /* MONITOR normal return status */
974 1
975 1 Declare
976 1 CDBHEAD GLOBALREF; /* Table of CDB's */
977 1 2 CDBLOCK (0:127) CHAR(CDB$K_SIZE);
978 1
979 1
```


EXTERNAL ROUTINE DEFINITIONS

```
980 1 /*
981 1 /*
982 1 /*
983 1 /*
984 1 /*
985 1 /*
986 1 /*/
987 1
988 1 Declare
989 1 EXECUTE_REQUEST ENTRY RETURNS(FIXED BINARY(31)), /* MONITOR routine to execute a MONITOR request */
990 1 ESTAB_CTRLCH ENTRY RETURNS(FIXED BINARY(31)), /* MONITOR MACRO-32 routine to set up CTRL-C and CTR
991 1 INPUT_INIT ENTRY RETURNS(FIXED BINARY(31)), /* MONITOR routine to do initialization on input fil
992 1 INPUT_CLEANUP ENTRY RETURNS(FIXED BINARY(31)), /* MONITOR routine to do cleanup on input file */
993 1 SUMMARY_INIT ENTRY RETURNS(FIXED BINARY(31)), /* MONITOR MACRO-32 routine to do summary init */
994 1 SUMMARY_CLEANUP ENTRY RETURNS(FIXED BINARY(31)), /* MONITOR routine to do cleanup on summary file */
995 1 MON_ERR ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE), /* MONITOR MACRO-32 routine to log synchronous error
996 1 SIGNAL_MON_ERR ENTRY, /* MONITOR MACRO-32 routine to signal MONITOR errors
997 1 LINK_MON_ERR ENTRY (ANY VALUE, ANY VALUE), /* MONITOR MACRO-32 routine to link a MONITOR error
998 1 QUAD_LT_QUAD ENTRY (BIT(64) ALIGNED, BIT(64) ALIGNED) /* MONITOR MACRO-32 unsigned quadword compare routin
999 1 RETURNS(BIT(1));
1000 1
1001 1
1002 1
1003 1 /*
1004 1 /*
1005 1 /*
1006 1 /*
1007 1 /*
1008 1 /*
1009 1 /*/
1010 1
1011 1 Declare
1012 1 MNR$_NOCLASS FIXED BINARY(31) GLOBALREF VALUE,
1013 1 MNR$_CLASDISAB FIXED BINARY(31) GLOBALREF VALUE,
1014 1 MNR$_NOCOMMSTLEV FIXED BINARY(31) GLOBALREF VALUE,
1015 1 MNR$_PREMEOF FIXED BINARY(31) GLOBALREF VALUE,
1016 1 MNR$_INVINPFIL FIXED BINARY(31) GLOBALREF VALUE,
1017 1 MNR$_NOCLASSES FIXED BINARY(31) GLOBALREF VALUE,
1018 1 MNR$_NOINPFILES FIXED BINARY(31) GLOBALREF VALUE,
1019 1 MNR$_IGNFIL FIXED BINARY(31) GLOBALREF VALUE,
1020 1 MNR$_NOSUMM FIXED BINARY(31) GLOBALREF VALUE,
1021 1 MNR$_DISPERR FIXED BINARY(31) GLOBALREF VALUE;
1022 1
1023 1
```

MESSAGE DEFINITIONS

```
1024 | 1 | /*
1025 | 1 | /*
1026 | 1 | /*
1027 | 1 | /*
1028 | 1 | /*
1029 | 1 | /*
1030 | 1 | /*
1031 | 1 |
1032 | 1 | Declare
1033 | 1 | MFSPTR          POINTER GLOBALDEF,          /* Pointer to Multi-File Summary Block (MFS) */
1034 | 1 | CSBVEC_PTR      POINTER GLOBALDEF,          /* Pointer to CSB Vector */
1035 | 1 | 1 CSBVEC,        /* CSB Vector (of ptrs to CSBs) */
1036 | 1 | 2 CSB_POINTER (1:MAX_INP_FILES) POINTER;    /* CSB pointers */
1037 | 1 |
1038 | 1 | /*
1039 | 1 | /*
1040 | 1 | /*
1041 | 1 | /*
1042 | 1 | /*
1043 | 1 | /*
1044 | 1 | /*
1045 | 1 |
1046 | 1 | %REPLACE      NOT_SUCCESSFUL      BY '0'B;          /* Failing status bit */
1047 | 1 | %REPLACE      YES                  BY '1'B;          /* For general use */
1048 | 1 | %REPLACE      NO                   BY '0'B;          /* For general use */
1049 | 1 | %REPLACE      AND_OP               BY '0001'B;       /* AND Boolean operation */
1050 | 1 | %REPLACE      OR_OP                BY '0111'B;       /* OR Boolean operation */
1051 | 1 |
```



```
1052 1  /*
1053 1  /*
1054 1  /*
1055 1  /*
1056 1  /*
1057 1  /*
1058 1  /*/
1059 1
1060 1  Declare
1061 1  CALL          FIXED BINARY(31),
1062 1  STATUS        BIT(1)  BASED(ADDR(CALL));
1063 1
1064 1  Declare
1065 1  IFBPTR        POINTER,
1066 1  CSBPTR        POINTER,
1067 1  DCDB          POINTER STATIC;
1068 1
1069 1  Declare
1070 1  IFB_TAB_PTR   POINTER,
1071 1  1 IFB_TABLE    BASED(IFB_TAB_PTR),
1072 1  2 AN_IFB       (1:MAX_INP_FILES) CHAR(IFB$K_SIZE);
1073 1
1074 1  Declare
1075 1  01 CURR_CLASS_DESCR (MAX_CLASS_NO+1),
1076 1
1077 1
1078 1  02 CURR_CDBPTR  POINTER,
1079 1  02 CURR_CLASS_NO FIXED BINARY(7);
1080 1
1081 1  Declare
1082 1  1 BUFFERS       BASED(MFSSA_STATSBUF),
1083 1  2 TOT (1:MAXELTS_MFS) FIXED BINARY(31),
1084 1  2 MIN (1:MAXELTS_MFS) FIXED BINARY(31),
1085 1  2 MAX (1:MAXELTS_MFS) FIXED BINARY(31);
1086 1
1087 1  Declare
1088 1  I              FIXED BINARY(15),
1089 1  J              FIXED BINARY(15),
1090 1  NON_EMPTY_FILES FIXED BINARY(15),
1091 1  ST_LEV_COMM     CHAR(8),
1092 1  FILE_CLASSES    BIT(128) ALIGNED,
1093 1  ALL_FILE_CLASSES BIT(128) ALIGNED,
1094 1  CURR_NODENAME    CHAR(16),
1095 1  FOUND_NODE       BIT(1) ALIGNED,
1096 1  CURR_ERRCODE     FIXED BINARY(31),
1097 1  MF_REQ_STATUS    FIXED BINARY(31),
1098 1  ALREADY_FAILED  BIT(1) ALIGNED;
1099 1
```

OWN STORAGE

```
/* Holds function value (return status) of called ro
/* Low-order status bit for called routines */

/* Pointer to Input File Block (IFB) */
/* Pointer to Column Summary Block (CSB) */
/* CDB for current class */

/* Pointer to IFB_TABLE */
/* Input File Block (IFB) Table */
/* A single IFB */

/* Current Class Descriptor */
/* This array of structures includes a CCD (Current
/* Class Descriptor) for each possible class. */
/* CDBPTR for current class */
/* Class number for current class */

/* Statistics buffers */
/* TOTAL statistics buffer */
/* MINIMUM statistics buffer */
/* MAXIMUM statistics buffer */

/* Loop control */
/* Loop control */
/* Count of non-empty input files */
/* Common file structure level */
/* Class bit string from current input file */
/* Class bit string which is union (OR) of all input
/* Nodename from current input file */
/* YES => a matching node was found in a CSB */
/* MONITOR error status code currently expected */
/* MFSUM REQUEST status code */
/* YES => a failure has already been signaled */
```

```
1100 1 ON FINISH; /* On finish, do nothing */
1101 1 ALREADY_FAILED = NO; /* Indicate no failure yet signaled */
1102 1 CURR_ERRCODE = 0; /* Set expected MONITOR code to default */
1103 1
1104 1 /*
1105 1 /* Set up condition handler to terminate the MONITOR request on:
1106 1 /* 1) any asynchronous error condition, such as file and I/O errors;
1107 1 /* 2) any synchronous MONITOR-detected condition.
1108 1 /*
1109 1
1110 1 ON ANYCONDITION /* On any condition signaled, */
1111 1 BEGIN;
1112 2
1113 2 Declare
1114 2 MNR$ERRINPFIL FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
1115 2 MNR$ERRRECFIL FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
1116 2 MNR$UNEXPERR FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
1117 2 MON_CODE FIXED BINARY(31), /* Monitor message code */
1118 2 TEMP FIXED BINARY(31), /* Temporary scratch area */
1119 2 MNR$FACNO FIXED BINARY(31) GLOBALREF VALUE, /* MONITOR facility number */
1120 2 ON_FILE CHAR(100) VARYING, /* Holds possible file name string */
1121 2 SIGNED_ERR_ENTRY (ANY VALUE, ANY VALUE, ANY VALUE, ANY); /* Rtn to set up PUTMSGVEC */
1122 2
1123 2 IF ^ALREADY_FAILED /* If a failure not already signaled, */
1124 2 THEN DO;
1125 3 ALREADY_FAILED = YES; /* Indicate a failure has been signaled */
1126 3 CHFSARGPTR = ONARGSLIST(); /* Get signal array pointer */
1127 3 ST$VALUE = CHFS$SIG_NAME; /* Get code for signaled condition */
1128 3 UNSPEC(TEMP) = ST$FAC_NO; /* Convert facility no. to binary in TEMP */
1129 3 IF TEMP = MNR$FACNO /* If a MONITOR code, */
1130 3 THEN MON_CODE = ST$VALUE; /* then remember it */
1131 3 ELSE DO; /* Otherwise, need to set the MON_CODE */
1132 4 ON_FILE = ONFILE(); /* Get PL/I file constant if I/O cond */
1133 4 IF ON_FILE = 'INPUT_FILE' /* If input file error, */
1134 4 THEN MON_CODE = MNR$ERRINPFIL; /* Set Monitor status code accordingly */
1135 4 ELSE IF CURR_ERRCODE = 0 /* Else, see if an error is currently expected */
1136 4 THEN MON_CODE = MNR$UNEXPERR; /* No, set "unexpected" code */
1137 4 ELSE MON_CODE = CURR_ERRCODE; /* Yes, set currently expected code */
1138 4 CURR_ERRCODE = 0; /* Reset to default MONITOR error code ("unexpected" */
1139 4 CALL SIGNED_ERR(MON_CODE, ST$VALUE, DIM(CHFS$SIG_ARG, 1), CHFS$SIG_ARG); /* Log the error */
1140 4 END;
1141 3
1142 3 MF_REQ_STATUS = MON_CODE; /* Set up code for MONITOR request termination */
1143 3 CALL MF_REQ_CLEANUP(); /* Perform cleanup for files, memory, etc. */
1144 3 END;
1145 2
1146 2 GO TO MF_REQ_EXIT; /* Go return from MFSUM_REQUEST (PL/I does an UNWIND */
1147 2
1148 2 END; /* End of ON-condition routine */
1149 1
```



```
1150 1 MC->MCASV_EOF = NO; /* End-of-file not hit yet */
1151 1 ON ENDFILE(INPUT_FILE) MC->MCASV_EOF = YES; /* Set up EOF condition */
1152 1
1153 1 CALL = MFSUM_INIT(); /* Perform init for this m.f. summary request */
1154 1 IF STATUS = NOT_SUCCESSFUL THEN CALL SIGNAL_MON_ERR(); /* Signal error if failure */
1155 1
1156 1 /*
1157 1 /* Establish CTRL-C and CTRL-Z handlers for terminating the MONITOR request.
1158 1 /* CTRL-C causes a MONITOR> prompt. CTRL-Z returns to DCL.
1159 1 /*
1160 1
1161 1 CALL = ESTAB_CTRLZ(); /* Establish CTRL-C and CTRL-Z handlers */
1162 1 /* If error, do not terminate; simply ignore CTRL-C'
1163 1
1164 1 /*
1165 1 /* For each input file, open it, do some checking of structure level, and classes,
1166 1 /* and build the Column Summary Block (CSB) (if necessary) to represent the column
1167 1 /* required in the summary report. There will be one column per file, or, if a
1168 1 /* by-node report was requested, one column per node.
1169 1 /*
1170 1
1171 1 NON_EMPTY_FILES = 0; /* Init count of non-empty input files */
1172 1 DO I = 1 TO M->MRBSB_INP_FILES WHILE(CTRLCZ_HIT = NO); /* Loop once per input file */
1173 2 /* ... as long as user has not terminated */
1174 2 IFBPTR = ADDR(AN_IFB(I)); /* Establish current Input File Block */
1175 2 M->MRBSA_INPUT = IFBSA_INPUT; /* Establish current input file */
1176 2 MC->MCASV_EOF = NO; /* End-of-file not hit yet */
1177 2
1178 2 CALL = INPUT_INIT(); /* Open input file and establish H as header record
1179 2 IF STATUS = NOT_SUCCESSFUL THEN /* If failed, */
1180 2 IF CALL ^= MNR$_PREMEOF THEN CALL SIGNAL_MON_ERR(); /* then signal error for all but PREMEOF (empty fi
1181 2 ELSE DO; /* but for PREMEOF, */
1182 2 IFBSB_COL_NO = 0; /* zero out column number, */
1183 2 CALL LINK_MON_ERR(MNR$_IGNFIL,IFBSA_INPUT); /* ... and issue info msg */
1184 2 END;
1185 2
1186 2 ELSE /* If successful, */
1187 2 DO; /* Beginning of successful do-group */
1188 2 NON_EMPTY_FILES = NON_EMPTY_FILES + 1; /* Count this good file */
1189 2 IF NON_EMPTY_FILES = 1 /* If first file, */
1190 2 THEN ST_CEV_COMM = H->MNR_HDR$T_LEVEL; /* ... then establish common structure level */
1191 2 ELSE IF ST_CEV_COMM ^= H->MNR_HDR$T_LEVEL /* ... otherwise, file level must match common */
1192 2 THEN DO;
1193 2 CALL MON_ERR(MNR$_NOCOMMSTLEV); /* ... if not, it's an error */
1194 2 CALL SIGNAL_MON_ERR(); /* ... signal it */
1195 2 END;
1196 2
1197 2 IF MNR_HDR$K_CLASSBITS < MC->MCASL_INPUT_LEN /* If CLASSBITS field is defined for input file, */
1198 2 THEN FILE_CLASSES = H->MNR_HDR$O_CLASSBITS; /* then get file classes from usual place */
1199 2 ELSE FILE_CLASSES = H->MNR_HDR$O_REVOCLSBITS; /* else get them from another place */
1200 2 /* NOTE -- MNR_HDR$O_REVOCLSBITS is used for compati
1201 2 /* with MONSC001 and MONBA001 file struct le
1202 2
1203 2 ALL_FILE_CLASSES = BOOL(ALL_FILE_CLASSES,FILE_CLASSES,OR_OP); /* Augment class string with class bits from this fi
1204 2
1205 2 CALL = GET_NODENAME(); /* Get node name from current file */
```

F 4
16-SEP-1984 02:18:13 VAX-11 PL/I X2.1-273 Page 10
5-SEP-1984 15:09:17 DISK\$VMSMASTER:[MONITOR.SRC]MF\$SUMM.PLI;1 (8)

```
IF STATUS = NOT_SUCCESSFUL THEN CALL SIGNAL_MON_ERR();      /* Signal error if failure */
```



```
1208 : 3      /*
1209 : 3      /* Set up Column Summary Block (CSB)
1210 : 3      /*/
1211 : 3
1212 : 3      IF ^ M->MRBSV_BY_NODE : NON_EMPTY_FILES = 1      /* If not by-node request, or first input file */
1213 : 3      THEN CALL CREATE_NEW_CSB();      /* then create a new CSB for a new column */
1214 : 3      ELSE DO;      /* else go look for a column with same node */
1215 : 4      /* name as input file */
1216 : 4      FOUND_NODE = NO;      /* Indicate haven't found a matching node yet */
1217 : 4      DO J = 1 TO MFS$B_COLUMNS WHILE (FOUND_NODE = NO);      /* Loop once per CSB until found one */
1218 : 5      CSBPTR = CSB_POINTER(J);      /* Get next CSB pointer */
1219 : 5      IF CURR_NODENAME = CSB$N_NODENAME      /* If CSB node same as input file node, */
1220 : 5      THEN DO;      /* Stash column no. in IFB for later use */
1221 : 6      IFB$B_COL_NO = J;      /* Indicate found a matching node */
1222 : 6      FOUND_NODE = YES;
1223 : 6      END;
1224 : 5      END;
1225 : 4      IF FOUND_NODE = NO      /* If no matching node found in all CSBs, */
1226 : 4      THEN CALL CREATE_NEW_CSB();      /* then create a new one */
1227 : 4      END;      /* End of successful do-group */
1228 : 3      END;
1229 : 2
1230 : 2      /*
1231 : 2      /* Perform cleanup operations on this input file (close it and free buffer).
1232 : 2      /*/
1233 : 2
1234 : 2      IF M->MRBSV_INP_CL_REQ      /* If cleanup required for this file, */
1235 : 2      THEN CALL =INPUT_CLEANUP();      /* ... then do cleanup for it */
1236 : 2
1237 : 2      END;
1238 : 1
1239 : 1      IF NON_EMPTY_FILES = 0      /* If all files are empty, */
1240 : 1      THEN DO;      /* then log the error */
1241 : 2      CALL MON_ERR(MNRS_NOINPFILES);      /* ... and signal it */
1242 : 2      CALL SIGNAL_MON_ERR();
1243 : 2      END;
1244 : 1
1245 : 1      /*
1246 : 1      /* Eliminate unnecessary classes and set up Current Class Descriptor Array
1247 : 1      /*/
1248 : 1
1249 : 1      IF CTRLCH_HIT = NO      /* If request is still running, */
1250 : 1      THEN DO;      /* Eliminate requested classes not in input */
1251 : 2      MFS$O_CLASSBITS = BOOL(ALL_FILE_CLASSES,MFS$O_CLASSBITS,AND_OP);      /* If no classes left to do, */
1252 : 2      IF MFS$O_CLASSBITS = '0'B      /* then log error */
1253 : 2      THEN DO;      /* ... and signal it */
1254 : 3      CALL MON_ERR(MNRS_NOCLASS);
1255 : 3      CALL SIGNAL_MON_ERR();
1256 : 3      END;
1257 : 2
1258 : 2
1259 : 2      /* NOTE ***** steal code to print warning message if one or more classes missing from the host of input file
1260 : 2
1261 : 2      /*
1262 : 2      /* Establish Current Class Descriptor Array
1263 : 2      /*/
```

MFSUM REQUEST
V04-000

H 4
16-SEP-1984 02:18:15
5-SEP-1984 15:09:17

VAX-11 PL/I X2.1-273 Page 12
DISK\$VMSMASTER:[MONITOR.SRC]MFSUMM.PLI;1 (9)

1264 2
1265 2
1266 2
1267 2
1268 2
1269 1

CALL = ESTAB_CCD();
IF STATUS = NOT_SUCCESSFUL THEN CALL SIGNAL_MON_ERR();
END;

/* Set up CCD */
/* Signal error if failure */


```
1270 1
1271 1 /*
1272 1 /* Main processing loop. For each input file, call EXECUTE REQUEST
1273 1 /* to perform the MONITOR request for that file. After each request
1274 1 /* has terminated, the SUM buffer for each class will have been stored
1275 1 /* in the multi-file summary buffer.
1276 1 /* Then, when all input files have been processed, execute MF_SUMMARY_EVENT
1277 1 /* to transform the multi-file summary buffer into the summary report.
1278 1 /*/
1279 1
1280 1 /*
1281 1 /* Temporarily de-establish CTRL/C and CTRL/Z handlers, since they will be
1282 1 /* established during each call to EXECUTE_REQUEST.
1283 1 /*/
1284 1
1285 1 IF CTRLZ_HIT = NO & CTRLZ_CHAN ^= 0 /* If request is still running, */
1286 1 THEN DO;
1287 2 CALL = SYSSCANCEL(CTRLZ_CHAN); /* Cancel CTRL-C and CTRL-Z handlers */
1288 2 CTRLZ_CHAN = 0; /* ... and indicate their disestablishment */
1289 2 END;
1290 1
1291 1 DO I = 1 TO M->MRBSB_INP_FILES WHILE(CTRLZ_HIT = NO); /* Loop once for each input file */
1292 2 IFBPTR = ADDR(AN_IFB(I)); /* ... as long as user has not killed request */
1293 2 IF IFBSB_COL_NO ^= 0 /* Get next IFB pointer */
1294 2 THEN DO; /* If this is a non-empty file, */
1295 3 MFSSB_CUR_COL = IFBSB_COL_NO; /* Set up current column no. for use by EXEC
1296 3 M->MRBSA_INPUT = IFBSA_INPUT; /* Set up MRB to refer to current input file
1297 3 M->MRBSQ_BEGINNING = MFSSQ_BEGINNING; /* Set up requested beginning time */
1298 3 M->MRBSQ_ENDING = MFSSQ_ENDING; /* Set up requested ending time */
1299 3 M->MRBSQ_CLASSBITS = MFSSQ_CLASSBITS; /* Start off each file with requested classe
1300 3
1301 3 CALL = EXECUTE_REQUEST(); /* Execute MONITOR request for this input fi
1302 3 IF STATUS = NOT SUCCESSFUL /* If request failed, */
1303 3 THEN CALL LINK_MON_ERR(MNRS_IGNFIL,IFBSA_INPUT); /* Let user know we ignored a file */
1304 3
1305 3 ELSE DO; /* Request successful ... check begin and en
1306 4 CSBPTR = CSB_POINTER(IFBSB_COL_NO); /* Get CSB pointer for current column */
1307 4 CSBSB_FILES = CSBSB_FILES + 1; /* Count this file in this column */
1308 4 IF ^ M->MRBSV_BY_NODE : QUAD LT QUAD(M->MRBSQ_BEGINNING,CSBSQ_BEGINNING) /* If new beginning establish
1309 4 THEN CSBSQ_BEGINNING = M->MRBSQ_BEGINNING; /* ... store it into CSB */
1310 4 IF ^ M->MRBSV_BY_NODE : QUAD LT QUAD(CSBSQ_ENDING,M->MRBSQ_ENDING) /* If new ending established, */
1311 4 THEN CSBSQ_ENDING = M->MRBSQ_ENDING; /* ... store it into CSB */
1312 4 END;
1313 4 END;
1314 3 END;
1315 2 END;
1316 1
```

```
1317 1 IF CTRLZ_HIT = NO /* If user hasn't hit CTRL/C or /Z, */
1318 1 THEN CALL = ESTAB_CTRLZ(); /* then re-establish CTRL-C and CTRL-Z handlers */
1319 1 /* If error, do not terminate; simply ignore CTRL-C'
1320 1 /*
1321 1 /* Summarize all accumulated data into a single report.
1322 1 /*/
1323 1
1324 1 IF CTRLZ_HIT = NO /* If user is still patient, */
1325 1 THEN DO;
1326 2 CALL = MF_SUMMARY_EVENT(); /* Output entire summary buffer for all classes */
1327 2 IF STATUS = NOT_SUCCESSFUL THEN CALL SIGNAL_MON_ERR(); /* Signal error if failure */
1328 2 END;
1329 1
1330 1 /*
1331 1 /* Perform cleanup operations for this M.F. Summary request
1332 1 /*/
1333 1
1334 1 CALL MF_REQ_CLEANUP(); /* Execute various cleanup routines */
1335 1
1336 1 /*
1337 1 /* Exit from MFSUM REQUEST routine.
1338 1 /* Note -- we get to this point either by falling through
1339 1 /* the above code (normal path), or by direct branch from
1340 1 /* the condition-handling routine (error path).
1341 1 /*/
1342 1
1343 1 MF_REQ_STATUS = NORMAL; /* Normal status if we get to this point */
1344 1
1345 1 MF_REQ_EXIT:
1346 1
1347 1 RETURN(MF_REQ_STATUS); /* Return to MONMAIN with completion status */
1348 1
1349 1
```



```
1350 1 MFSUM_INIT: Procedure Returns(Fixed Binary(31)); /* Initialization for M.F. Summary Request */
1351 2
1352 2 Declare
1353 2 CLASSES BIT(128); /* Temporary class bit string */
1354 2 CLASSES_VEC (0:127) BIT(1) DEFINED(CLASSES); /* Bit-addressable alias */
1355 2
1356 2 CTRLCZ_HIT = NO; /* Indicate CTRL-C and CTRL-Z not hit */
1357 2 CTRLCZ_CHAN = 0; /* ... and no channel assigned for them */
1358 2
1359 2 CSBVEC_PTR = ADDR(CSBVEC); /* Stash address of CSB Vector for later use */
1360 2
1361 2 ALLOCATE MFS; /* Allocate space for Multi-File Summary block */
1362 2
1363 2 MFSSB_COLUMNS = 0; /* Start out request with no columns */
1364 2
1365 2 ALL_FILE_CLASSES = '0'B; /* Start out request with no input file classes */
1366 2
1367 2 /*
1368 2 /* Save requested BEGINNING and ENDING times in the MFS so they
1369 2 /* can later be re-loaded to the MRB as EXECUTE_REQUEST is called
1370 2 /* for each input file.
1371 2 /*/
1372 2
1373 2 MFSSQ_BEGINNING = M->MRBSQ_BEGINNING; /* Save beginning time */
1374 2 MFSSQ_ENDING = M->MRBSQ_ENDING; /* Save ending time */
1375 2
1376 2 MFSSA_IFB_TAB = M->MRBSA_INPUT; /* Save pointer to IFB Table for later use */
1377 2 IFB_TAB_PTR = MFSSA_IFB_TAB; /* Set up IFB Table */
1378 2
1379 2 /*
1380 2 /* Obtain space for statistics buffers
1381 2 /*/
1382 2
1383 2 ALLOCATE BUFFERS; /* Get the space and set up MFSSA_STATSBUF */
1384 2 MFSSL_STATSBUF = 4 * 3 * MAXELTS_MFS; /* Compute and save its length (in bytes) */
1385 2
1386 2 /*
1387 2 /* Turn off class bits for classes not eligible for
1388 2 /* multi-file summary.
1389 2 /*/
1390 2
1391 2 CLASSES = M->MRBSQ_CLASSBITS; /* Get list of requested classes */
1392 2 CLASSES_VEC(PROCS_CLSNO) = NO; /* Turn off PROCESSES class bit */
1393 2
1394 2 MFSSQ_CLASSBITS = CLASSES; /* Move updated class bit string to the MFS */
1395 2
1396 2 IF MFSSQ_CLASSBITS = '0'B /* If no classes left to do, */
1397 2 THEN DO; /*
1398 2 CALL MON_ERR(MNR$NOCLASSES); /* then log error */
1399 2 RETURN(MNR$NOCLASSES); /* ... and return with status */
1400 2 END;
1401 2
```

```

1402 | 2 /*
1403 | 2 /* Check for /SUMMARY qualifier specified, and return
1404 | 2 /* error if not. Move summary file spec ptr from MRB to
1405 | 2 /* MFS, and explicitly clear the /SUMMARY, /DISPLAY
1406 | 2 /* and /RECORD file spec ptrs from the MRB.
1407 | 2 /*/
1408 | 2
1409 | 2 IF M->MRB$A_SUMMARY = NULL()
1410 | 2 THEN DO;
1411 | 2 CALL MON ERR(MNR$ NOSUMM);
1412 | 2 RETURN(MNR$_NOSUMM);
1413 | 2 END;
1414 | 2 ELSE DO;
1415 | 2 MFSSA_SUMMARY = M->MRB$A_SUMMARY;
1416 | 2 M->MRB$A_SUMMARY = NULL();
1417 | 2 M->MRB$A_DISPLAY = NULL();
1418 | 2 M->MRB$A_RECORD = NULL();
1419 | 2 END;
1420 | 2
1421 | 2 RETURN(NORMAL);
1422 | 2 END MFSUM_INIT;
1423 | 1

```

```

/* If no summary qualifier, */
/* then log the error */
/* ... and return with status */

/* Move /SUMMARY file spec ptr to MFS */
/* Indicate no regular /SUMMARY */
/* Indicate no /DISPLAY */
/* Indicate no /RECORD */

/* Return */

```



```

1424 1 CREATE_NEW_CSB: Procedure;
1425 2
1426 2 MFSSB_COLUMNS = MFSSB_COLUMNS + 1;
1427 2 IFBSB_COL_NO = MFSSB_COLUMNS;
1428 2
1429 2 ALLOCATE CSB;
1430 2 CSB_POINTER(MFSSB_COLUMNS) = CSBPTR;
1431 2 CSBST_NODENAME = CURR_NODENAME;
1432 2 CSBSQ_BEGINNING = 'FFFFFFFFFFFFFFFF'B4;
1433 2 CSBSQ_ENDING = '0'B;
1434 2 CSBSB_FILES = 0;
1435 2 CSBSV_IGNORE = NO;
1436 2
1437 2 RETURN;
1438 2 END CREATE_NEW_CSB;
1439 1

```

```

/* Create a new Column Summary Block */
/* Count a new column */
/* ... and store its number in current IFB */
/* Get space for CSB */
/* Add its pointer to CSB vector */
/* Get nodename from current file */
/* Init to latest beginning time */
/* ... and earliest ending time */
/* Init no. of files in this column */
/* Don't ignore this column */
/* Return */

```

```
1440 1 GET_NODENAME: Procedure Returns(Fixed Binary(31));          /* Get nodename from current input file */
1441 2
1442 2 Declare
1443 2 READ_INPUT          ENTRY (FIXED BINARY(31)),                /* MONITOR routine to read an input (playback) file */
1444 2 NEXT_REC            FIXED BINARY(31) GLOBALREF VALUE;        /* Read next record indicator for READ_INPUT rtn */
1445 2
1446 2 Declare
1447 2 SYI_TYPE            FIXED BINARY(15) GLOBALREF,                /* Type for MONITOR recording file sys info record */
1448 2 TEMP_TYPE           BIT(8) ALIGNED,                            /* Temporary area for record type byte */
1449 2 SYIPTR              POINTER;                                    /* Pointer to sys info record */
1450 2
1451 2 CALL READ_INPUT(NEXT_REC);                                     /* Read system information record */
1452 2 IF MC->MC$V_EOF                                             /* If end-of-file, */
1453 2 THEN DO;
1454 2     CALL MON ERR(MNRS_PREMEOF);                                /* Can't find sys info record; log the error */
1455 2     RETURN (MNRS_PREMEOF);                                    /* ... and return to caller */
1456 2 END;
1457 2
1458 2 SYIPTR = MC->MC$A_INPUT_PTR;                                  /* Establish ptr to sys info record */
1459 2 TEMP_TYPE = UNSPEC(SYI_TYPE);                                /* Get sys info type into a byte for compare */
1460 2 IF SYIPTR->MNR_SYISB_TYPE ^= TEMP_TYPE                       /* If this record is not the sys info rec, */
1461 2 THEN DO;
1462 2     CALL MON ERR(MNRS_INVINPFIL);                                /* Log an error */
1463 2     RETURN(MNRS_INVINPFIL);                                    /* ... and return to caller */
1464 2 END;
1465 2
1466 2 IF MNR_SYISK_NODENAME < MC->MC$A_INPUT_LEN                    /* If NODENAME field is defined for input file, */
1467 2 THEN CURR_NODENAME = SYIPTR->MNR_SYIST_NODENAME;            /* ... then pick it up from there */
1468 2 ELSE UNSPEC(CURR_NODENAME) = '0'B;                            /* Otherwise, simply clear it */
1469 2
1470 2 RETURN(NORMAL);
1471 2 END GET_NODENAME;
1472 1
```



```
1473 1 ESTAB_CCD: Procedure Returns(Fixed Binary(31));          /* Establish Current Class Descriptor array */
1474 2
1475 3 /*
1476 4 /*
1477 5 /*
1478 6 /*
1479 7 /*
1480 8 /*
1481 9 /*
1482 10 /*
1483 11
1484 12 DO_CLASSES BIT(128),          /* Classes to do */
1485 13 DO_CLASSES_VEC (0:127) BIT(1) DEFINED(DO_CLASSES),      /* Bit-addressable alias */
1486 14 CLASS_NO FIXED BINARY(7);    /* Class number */
1487 15
1488 16 /*
1489 17 /* Given MFS$O CLASSBITS, execute do loop using INDEX builtin
1490 18 /* to fill in the CCD (Current Class Descriptor) array.
1491 19 /*
1492 20
1493 21 DO_CLASSES = MFS$O_CLASSBITS;          /* Get list of classes to do */
1494 22 CLASS_NO = 0;                          /* Initialize class number */
1495 23 DO I = 1 TO MAX CLASS_NO + 1 WHILE(CLASS_NO >= 0);        /* Loop once for each possible class */
1496 24 CLASS_NO = INDEX(DO_CLASSES,YES) - 1;    /* Find next requested class number */
1497 25 IF CLASS_NO >= 0          /* Only continue if a class was found */
1498 26 THEN DO;
1499 27 DO_CLASSES_VEC(CLASS_NO) = NO;          /* Eliminate it from future consideration */
1500 28 CURR_CLASS_NO(I) = CLASS_NO;          /* Store class_no in CCD table */
1501 29 MFS$O_CLASSCT = I;                    /* Keep track of class count */
1502 30 CURR_CDBPTR(I) = ADDR(CDBLOCK(CLASS_NO));    /* ... and store it in CCD table */
1503 31 DCDB = CURR_CDBPTR(I);                /* Get CDB address'ty */
1504 32 DCDB->CDB$A_SUMBUF = NULL();          /* Make sure we get a new summary buffer */
1505 33 IF CURR_CDBPTR(I)->CDB$V_DISABLE = YES    /* If this class is disabled, */
1506 34 THEN DO;
1507 35 CALL MON_ERR(MNR$_CLASDISAB);          /* Log the error */
1508 36 RETURN(MNR$_CLASDISAB);              /* ... and return */
1509 37 END;
1510 38
1511 39 END;
1512 40
1513 41 RETURN(NORMAL);          /* Normal return */
1514 42
1515 43 END ESTAB_CCD;
1516 44
```

```

1517 1 MF_SUMMARY_EVENT: Procedure Returns(Fixed Binary(31)); /* Multi-File Summary Event */
1518 2
1519 2 /*
1520 2 /*++
1521 2 /*
1522 2 /* FUNCTIONAL DESCRIPTION:
1523 2 /*
1524 2 /* MF_SUMMARY_EVENT
1525 2 /*
1526 2 /* Called by MFSUM_REQUEST once per request to create a
1527 2 /* summary file containing one or more screen images for
1528 2 /* each of the requested classes.
1529 2 /*
1530 2 /* INPUTS:
1531 2 /*
1532 2 /* None
1533 2 /*
1534 2 /* OUTPUTS:
1535 2 /*
1536 2 /* None
1537 2 /*
1538 2 /* ROUTINE VALUE:
1539 2 /*
1540 2 /* SSS_NORMAL, or failing MONITOR status code.
1541 2 /*
1542 2 /*--
1543 2 /*/
1544 2

```



```
1545 | 2 /*
1546 | 2 /*
1547 | 2 /*
1548 | 2 /*
1549 | 2 /*
1550 | 2 /*
1551 | 2 /*
1552 | 2 /*
1553 | 2
1554 | 2 DECLARE
1555 | 2 CURR_CLASS          FIXED BINARY(15),
1556 | 2 FULL_PAGES         FIXED BINARY(7),
1557 | 2 PARTIAL_PAGE_COLS  FIXED BINARY(7),
1558 | 2 COLS_PER_PAGE      FIXED BINARY(7) INIT(5);
1559 | 2
1560 | 2 DECLARE
1561 | 2 LARGE_NO           FIXED BINARY(31) GLOBALREF VALUE,
1562 | 2 1 BUFFERS          BASED(MFSSA_STATSBUF),
1563 | 2 2 TOT (1:MAXELTS_MFS) FIXED BINARY(31),
1564 | 2 2 MIN (1:MAXELTS_MFS) FIXED BINARY(31),
1565 | 2 2 MAX (1:MAXELTS_MFS) FIXED BINARY(31);
1566 | 2
1567 | 2 DECLARE
1568 | 2 PUT_LEN            FIXED BINARY(31),
1569 | 2 1 DPUT_FLAGS,
1570 | 2 2 FAOL_REQUESTED BIT(8) ALIGNED,
1571 | 2 2 OUTP_REQUESTED BIT(8) ALIGNED;
1572 | 2
1573 | 2 DECLARE
1574 | 2 DISPLAY_PUT        ENTRY(ANY, FIXED BINARY(31), ANY, ANY)
1575 | 2                     OPTIONS(VARIABLE)
1576 | 2                     RETURNS(FIXED BINARY(31)),
1577 | 2 ADV_HOM_ITEM       ENTRY (POINTER);
1578 | 2
1579 | 2 DECLARE
1580 | 2 DCDB               POINTER STATIC;
1581 | 2
1582 | 2 DECLARE
1583 | 2 SPTR               POINTER GLOBALREF;
```

LOCAL STORAGE

```
/* Consec no (not class no) of current class */
/* Number of full pages to display */
/* Number of columns on the partial (last) page */
/* Number of columns which fit on a single page */

/* Very large number (integer or floating) */
/* Statistics buffers */
/* TOTAL statistics buffer */
/* MINIMUM statistics buffer */
/* MAXIMUM statistics buffer */

/* Length of buffer for DISPLAY_PUT to put */
/* DISPLAY_PUT routine flags */
/* YES => Xlate buffer with FAOL first */
/* YES => Really output buffer */

/* MACRO-32 rtn to put a display string */

/* MACRO-32 rtn to advance homog class to next displ

/* CDB for current class */

/* Pointer to SYI (System Information Area) */
```

```
1584 UNSPEC(SPTR->MNR_SYIST_NODENAME) = '0'B; /* Clear nodename to avoid misleading node in headin
1585
1586
1587 /*
1588 /* Loop through CSBs (Column Summary Blocks), turning on CSB$V_IGNORE
1589 /* for each one with CSB$Q_ENDING = 0. These are columns which never
1590 /* accumulated any data. If all columns are ignored, terminate the
1591 /* MONITOR request. Establish new value for MFSSB_COLUMNS which is
1592 /* the new column count, excluding ignored columns.
1593 /*/
1594
1595 J = 0; /* Init count of ignored columns */
1596 DO I = 1 TO MFSSB_COLUMNS; /* Loop once for each column */
1597 CSBPTR = CSB_POINTER(I); /* Get CSB pointer for this column */
1598 IF CSB$Q_ENDING = '0'B /* If no data for this column, */
1599 THEN DO;
1600 CSB$V_IGNORE = YES; /* ignore it, */
1601 J = J + 1; /* ... and count it */
1602 END;
1603 END;
1604
1605 MFSSB_COLUMNS = MFSSB_COLUMNS - J; /* Compute new column count */
1606 IF MFSSB_COLUMNS = 0 /* If no columns, */
1607 THEN DO;
1608 CALL MON_ERR(MNRS_NOINPFILES); /* then log the error */
1609 RETURN(MNRS_NOINPFILES); /* ... and return with status */
1610 END;
1611
1612 M->MRB$A_SUMMARY = MFSSA_SUMMARY; /* Get /SUMMARY file spec ptr to MRB for call */
1613 CALL = SUMMARY_INIT(); /* Do summary file init */
1614 IF STATUS = NOT_SUCCESSFUL /* Failed? */
1615 THEN DO;
1616 CALL MON_ERR(MNRS_DISPERR,CALL); /* Yes -- log the error */
1617 RETURN(MNRS_DISPERR); /* ... and return with status */
1618 END;
1619
1620 FULL_PAGES = DIVIDE(MFSSB_COLUMNS, COLS_PER_PAGE, 7); /* Compute no. of full report pages of data */
1621 PARTIAL_PAGE_COLS = MFSSB_COLUMNS - (FULL_PAGES * COLS_PER_PAGE); /* ... and number of partially filled pages */
1622
1623
```



```
1624 : 2      /*  
1625 : 2      /* Summarize each class individually.  
1626 : 2      /*/  
1627 : 2  
1628 : 2      DO CURR_CLASS = 1 TO MFSSW_CLASSCT;          /* Loop once for each requested class */  
1629 : 2  
1630 : 2      DCDB = CURR_CDBPTR(CURR_CLASS);              /* Get CDB ptr for current class */  
1631 : 2  
1632 : 2      IF DCDB->CDB$V_HOMOG                          /* If homogeneous class, */  
1633 : 2          THEN MFSSL_ELEMS = DCDB->CDB$A_CDX->CDX$SL_DCOUNT; /* then set this many elements */  
1634 : 2          ELSE IF DCDB->CDB$V_SYSCLS                /* else if SYSTEM class, */  
1635 : 2              THEN DO;                               /*  
1636 : 2                  MFSSL_ELEMS = ECOUNT_SYS_ALL;      /* then use SYS/ALL element count */  
1637 : 2                  DCDB->CDB$B_ST = ALL_STAT;        /* and force a tabular display */  
1638 : 2                  END;  
1639 : 2          ELSE MFSSL_ELEMS = DCDB->CDB$SL_ECOUNT; /* this many elements for other heteros */  
1640 : 2  
1641 : 2      IF DCDB->CDB$V_HOMOG                          /* If homogeneous class, */  
1642 : 2          THEN DO;                                   /*  
1643 : 2              DCDB->CDB$A_CDX->CDX$B_IDISCONSEC = 0; /* Init consec display item number */  
1644 : 2              DO WHILE(DCDB->CDB$A_CDX->CDX$B_IDISCONSEC < DCDB->CDB$A_CDX->CDX$B_IDISCT); /* Advance to next display item */  
1645 : 2                  CALL ADV_HOM_ITEM(DCDB);          /* Summarize once for each item */  
1646 : 2                  CALL = SUMM_ONE_CLASS();          /* Check call */  
1647 : 2                  IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);  
1648 : 2                  END;  
1649 : 2              END;  
1650 : 2  
1651 : 2          ELSE DO;                                    /* Heterogeneous class or PROCESSES */  
1652 : 2              CALL = SUMM_ONE_CLASS();                /* Only need to call once */  
1653 : 2              IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Check call */  
1654 : 2          END;  
1655 : 2  
1656 : 2      END;  
1657 : 2  
1658 : 2      RETURN(NORMAL);  
1659 : 2
```

/* Return */

```
1660 SUMM_ONE_CLASS: Procedure Returns(Fixed Binary(31));          /* Perform m.f. summary output for a single class */
1661
1662 /*
1663 /*++
1664 /*
1665 /* FUNCTIONAL DESCRIPTION:
1666 /*
1667 /*     SUMM_ONE_CLASS
1668 /*
1669 /*     Called by MF_SUMMARY_EVENT to put screen images to the
1670 /*     summary file for a single class.
1671 /*
1672 /* INPUTS:
1673 /*
1674 /*     None
1675 /*
1676 /* OUTPUTS:
1677 /*
1678 /*     None
1679 /*
1680 /* ROUTINE VALUE:
1681 /*
1682 /*     SSS_NORMAL, or failing MONITOR status code.
1683 /*
1684 /*--
1685
1686 /*/
1687 /*
1688 /*
1689 /*
1690 /*
1691 /*
1692 /*
1693 /*/
1694
1695 Declare
1696 PAGES          FIXED BINARY(15);          /* No. of full pages to display (loop control) */
1697
1698 Declare
1699 START_COL      FIXED BINARY(7);          /* Starting column number */
1700 COLS_TO_DO     FIXED BINARY(7);          /* Number of columns to display */
1701 END_COL        FIXED BINARY(7);          /* Ending column number */
1702 STAT_IND       BIT(1) ALIGNED;           /* Statistics Indicator -- YES => display statistics
1703
```



```
1704  
1705 :  
1706 :  
1707 :  
1708 :  
1709 :  
1710 :  
1711 :  
1712 :  
1713 :  
1714 :  
1715 :  
1716 :  
1717 :  
1718 :  
1719 :  
1720 :  
1721 :  
1722 :  
1723 :  
1724 :  
1725 :  
1726 :  
1727 :  
1728 :  
1729 :  
1730 :  
1731 :  
1732 :  
1733 :  
1734 :  
1735 :  
1736 :  
1737 :  
1738 :  
1739 :  
1740 :  
1741 :  
1742 :  
1743 :  
1744 :  
1745 :  
1746 :  
1747 :  
1748 :  
1749 :  
1750 :  
1751 :  
1752 :  
1753 :  
2
```

```
/*  
/* Initialize statistics (TOT, MIN, and MAX) buffers.  
*/  
  
DO I = 1 TO MFSSL_ELEMS; /* Loop once per element */  
TOT(I) = 0; /* Clear total to 0 */  
MIN(I) = LARGE_NO; /* Set min to largest number */  
MAX(I) = 0; /* Set max to smallest number */  
END;  
  
MFSSB_DATA_COLS = 0; /* Init no. of columns of data (used by FILL_MFSUM_F  
  
/*  
/* Put as many full pages of variable information to summary file as  
/* required by this class.  
*/  
  
START_COL = 1; /* Start with column 1 */  
  
DO PAGES = 1 TO FULL_PAGES; /* Loop once for each full page */  
CALL = PUT_SUMM_PAGE(START_COL, COLS_PER_PAGE, END_COL, NO); /* Put a single page for this class */  
IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */  
START_COL = END_COL + 1; /* Establish new starting column for next page */  
END;  
  
/*  
/* Now put one or, possibly, two partial pages of variable information  
/* to the summary file for this class.  
*/  
  
IF PARTIAL_PAGE_COLS <= 4 /* If no more than 4 columns on partial page, */  
THEN DO; /* ... then can fit the statistics on same page */  
CALL = PUT_SUMM_PAGE(START_COL, PARTIAL_PAGE_COLS, END_COL, YES); /* Put a single page for this class */  
IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */  
END;  
  
ELSE DO; /* ... otherwise, one page for data and one for stat  
CALL = PUT_SUMM_PAGE(START_COL, PARTIAL_PAGE_COLS, END_COL, NO); /* Put a single page for this class */  
IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */  
  
CALL = PUT_SUMM_PAGE(START_COL, 0, END_COL, YES); /* Put a single page for this class */  
IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */  
END;  
  
RETURN(NORMAL);  
  
END SUMM_ONE_CLASS;
```

```

1754 2 PUT_SUMM_PAGE: Procedure (START_COL, COLS_TO_DO, END_COL, STAT_IND) /* Output a single summary page for curr cla
1755 Returns(Fixed Binary(31));
1756
1757 /*
1758 /*++
1759 /*
1760 /* FUNCTIONAL DESCRIPTION:
1761 /*
1762 /* PUT_SUMM_PAGE
1763 /*
1764 /* TBS
1765 /*
1766 /* INPUTS:
1767 /*
1768 /* START_COL -- Column number of 1st column to display.
1769 /*
1770 /* COLS_TO_DO -- Number of columns to display.
1771 /*
1772 /* STAT_IND -- Flag indicating whether or not to display statistics.
1773 /* ON means "display statistics".
1774 /* OFF means "don't display statistics".
1775 /*
1776 /* OUTPUTS:
1777 /*
1778 /* END_COL -- Column number of last column displayed.
1779 /*
1780 /* ROUTINE VALUE:
1781 /*
1782 /* $$$_NORMAL, or failing MONITOR status code.
1783 /*
1784 /*--
1785 /*/
1786

```



```

1787  /*
1788  /*
1789  /*
1790  /*
1791  /*
1792  /*
1793  /*/
1794
1795  Declare
1796  START_COL      FIXED BINARY(7),          /* Starting column number */
1797  COLS_TO_DO     FIXED BINARY(7),          /* Number of columns to display */
1798  END_COL        FIXED BINARY(7),          /* Ending column number */
1799  STAT_IND       BIT(1) ALIGNED;           /* Statistics Indicator -- YES => display statistics
1800
1801  Declare
1802  DISP_TEMPLATE  ENTRY (POINTER, BIT(1) ALIGNED)          /* Rtn to display the summary template */
1803  RETURNS (FIXED BINARY(31)),
1804  FILL_MFSUM_FAOSTK ENTRY(POINTER, FIXED BINARY(7), FIXED BINARY(7), BIT(1) ALIGNED)
1805  RETURNS (FIXED BINARY(31)),          /* MACRO-32 rtn to fill the FAOSTK with data */
1806  DISPLAY_HOMOG  ENTRY (POINTER)          /* MACRO-32 rtn to display homog class data */
1807  RETURNS (FIXED BINARY(31));
1808
1809  Declare
1810  1 MF_SUMM1_STR GLOBALREF,          /* M.F. Summary fixed FAO control string */
1811  2 [             FIXED BINARY(7),   /* Length */
1812  2 S             CHAR(1);           /* First character of string */
1813
1814  Declare
1815  DATA_STR      CHAR(1) BASED(DCDB->CDB$A FAOCTR),      /* First char of FAO ctr str for summary data */
1816  FAOSTK         FIXED BINARY(31) GLOBALREF;              /* First longword of FAOL parm list */
1817
1818

```

```
1819  
1820 :  
1821 :  
1822 :  
1823 :  
1824 :  
1825 :  
1826 :  
1827 :  
1828 :  
1829 :  
1830 :  
1831 :  
1832 :  
1833 :  
1834 :  
1835 :  
1836 :  
1837 :  
1838 :  
1839 :  
1840 :  
1841 :  
1842 :  
1843 :  
1844 :  
1845 :  
1846 :  
1847 :  
1848 :  
1849 :  
1850 :  
1851 :  
1852 :  
1853 :  
1854 :  
1855 :  
1856 :  
1857 :  
1858 :  
1859 :  
1860 :  
1861 :  
1862 :  
1863 :  
1864 :  
1865 :  
1866 :  
1867 :  
1868 :  
1869 :  
1870 :  
1871 :  
1872 :  
1873 :  
1874 :  
3
```

```
/*  
/*      Fill FAOSTK with data for this page for this class  
*/  
  
CALL = FILL_MFSUM_FAOSTK(DCDB,START_COL,COLS_TO_DO,STAT_IND); /* Fill FAOSTK with data */  
IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */  
  
/*  
/*      Put out template, including utility identification, class name,  
/*      element labels for hetero classes, etc.  
*/  
  
CALL = DISP_TEMPLATE(DCDB,NO); /* Send template to SCRPKG, but don't output yet */  
IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Check call */  
  
/*  
/*      Put out column headings, including node name, beginning and  
/*      ending times, and statistics headings (TOT, AVE, MIN, MAX) if requested.  
*/  
  
/*  
/*      Start with fixed portions of "screen."  
*/  
  
PUT_LEN = MF_SUMM1_STR.L; /* Length of fixed summary control string */  
FAOL_REQUESTED = YES; /* Run it through FAOL */  
OUTP_REQUESTED = NO; /* ... but don't output it yet */  
CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,MF_SUMM1_STR.S.); /* Send summary line to SCRPKG */  
IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */  
  
/*  
/*      Next display node name column headings  
*/  
  
CALL = MFSUM_HEADINGS(START_COL,COLS_TO_DO,END_COL,STAT_IND); /* Display column headings */  
IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */  
  
/*  
/*      Put a screenful of data to summary file  
*/  
  
IF DCDB->CDB$V_HOMOG /* Check type of standard class */  
  THEN DO; /* Homogeneous Standard Class */  
    CALL = DISPLAY_HOMOG(DCDB); /* Send homog data display lines to SCRPKG */  
    IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */  
  END;  
  ELSE DO; /* Heterogeneous Standard Class */  
    FAOL_REQUESTED = YES; /* Run it through FAOL */  
    OUTP_REQUESTED = YES; /* Output it now */  
    CALL = DISPLAY_PUT(DPUT_FLAGS,DCDB->CDB$L_FAOCTR,DATA_STR,FAOSTK); /* Send display data to SCRPKG */  
    IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */  
  END;
```


MFSUM REQUEST
V04-000

L⁵
16-SEP-1984 02:18:22
5-SEP-1984 15:09:17

VAX-11 PL/I X2.1-273
DISK\$VMSMASTER:[MONITOR.SRC]MFSUMM.PLI;1 (25)

Page 29

1875 3
1876 3
1877 3
1878 2

RETURN(NORMAL);
END PUT_SUMM_PAGE;

/* Return */

```

1879      MFSUM_HEADINGS: Procedure (START_COL, COLS_TO_DO, END_COL, STAT_IND) /* Output column headings for curr class and page
1880      Returns(Fixed Binary(31));
1881
1882      /*
1883      /*++
1884      /*
1885      /* FUNCTIONAL DESCRIPTION:
1886      /*
1887      /*     MFSUM_HEADINGS
1888      /*
1889      /*     Send to SCRPKG a column heading for each of the requested columns
1890      /*     for the current output page and class. The heading consists of a
1891      /*     nodename and beginning and ending times.
1892      /*
1893      /* INPUTS:
1894      /*
1895      /*     START_COL  -- Column number of 1st column to display.
1896      /*
1897      /*     COLS_TO_DO -- Number of columns to display.
1898      /*
1899      /*     STAT_IND   -- Flag indicating whether or not to display statistics.
1900      /*                   ON means "display statistics headings".
1901      /*                   OFF means "don't display statistics headings".
1902      /*
1903      /* OUTPUTS:
1904      /*
1905      /*     END_COL    -- Column number of last column displayed.
1906      /*
1907      /* ROUTINE VALUE:
1908      /*
1909      /*     SS$_NORMAL, or failing MONITOR status code.
1910      /*
1911      /*--
1912      /*/
1913
1914

```


MFSUM REQUEST
V04-000

B 6
16-SEP-1984 02:18:23
5-SEP-1984 15:09:17

VAX-11 PL/I X2.1-273
DISK\$VMSMASTER:[MONITOR.SRC]MFSUMM.PLI;1 (27)

Page 32

1971	3	1	MFS_STHEAD2_STR GLOBALREF,
1972	3	2	L FIXED BINARY(7),
1973	3	2	S CHAR(1);
1974	3		
1975	3		

/* M.F. Summary stats head FAO control string (2nd)
/* Length */
/* First character of string */


```
1976 3 IF COLS_TO_DO ^= 0 /* If there is some work to do, */
1977 3 THEN DO;
1978 4
1979 4 /*
1980 4 /* Fill parameter lists for nodename and beginning and ending times.
1981 4 */
1982 4
1983 4 J = 0; /* Clear count of columns actually processed */
1984 4
1985 4 DO I = START_COL TO MAX_INP_FILES WHILE(J ^= COLS_TO_DO); /* Keep looking at columns until all processed */
1986 5 END_COL = I; /* Remember ending column number for caller */
1987 5 CSBPTR = CSB_POINTER(I); /* Get next CSB (Column Summary Block) ptr */
1988 5 IF CSB$V_IGNORE = NO /* If it is actually to be processed, */
1989 5 THEN DO;
1990 6 J = J + 1; /* Count this column */
1991 6 MFS_NODE_PARM(J) = ADDR(CSB$T_NODENAME); /* Move nodename parm to list */
1992 6 MFS_FILES_PARM(J) = CSB$B_FILES; /* Move "number of files" parm to list */
1993 6 IF CSB$B_FILES <= 1 /* If 1 or no files, */
1994 6 THEN MFS_FIELD_PARM(J) = 0; /* then field width = 0 (no display) */
1995 6 ELSE MFS_FIELD_PARM(J) = 5; /* otherwise, field width = 5 */
1996 6 MFS_BEG_PARM(J) = ADDR(CSB$Q_BEGINNING); /* Move beginning time parm to list */
1997 6 MFS_END_PARM(J) = ADDR(CSB$Q_ENDING); /* Move ending time parm to list */
1998 6 END;
1999 5
2000 4 END;
2001 4
2002 4 /*
2003 4 /* Put out node names.
2004 4 */
2005 4 PUT_LEN = COLS_TO_DO * MFS_NOD_SEGLEN; /* Compute length of control string */
2006 4 FAOL_REQUESTED = YES; /* Run it through FAOL */
2007 4 OUTP_REQUESTED = NO; /* ... but don't output it yet */
2008 4 CALL = DISPLAY_PUT(DPUT_FLAGS, PUT_LEN, MFS_NODE_STR.S, MFS_NODE_PARMS); /* Send all nodenames to SCRPKG */
2009 4 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2010 4
2011 4 /*
2012 4 /* Put out beginning times
2013 4 */
2014 4
2015 4 CALL SCR$SET_CURSOR(7,20); /* Explicitly set cursor */
2016 4 PUT_LEN = COLS_TO_DO * MFS_TIM_SEGLEN; /* Compute length of control string */
2017 4 FAOL_REQUESTED = YES; /* Run it through FAOL */
2018 4 OUTP_REQUESTED = NO; /* ... but don't output it yet */
2019 4 CALL = DISPLAY_PUT(DPUT_FLAGS, PUT_LEN, MFS_TIME_STR.S, MFS_BEG_PARMS); /* Send all beg times to SCRPKG */
2020 4 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2021 4
2022 4 /*
2023 4 /* Put out ending times
2024 4 */
2025 4
2026 4 CALL SCR$SET_CURSOR(8,20); /* Explicitly set cursor */
2027 4 CALL = DISPLAY_PUT(DPUT_FLAGS, PUT_LEN, MFS_TIME_STR.S, MFS_END_PARMS); /* Send all end times to SCRPKG */
2028 4 /* Note -- uses most arguments from previous call */
2029 4 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2030 4
2031 4 END;
```

MFSUM REQUEST
V04-000

D 6
16-SEP-1984 02:18:23
5-SEP-1984 15:09:17

VAX-11 PL/I X2.1-273
DISK\$VMSMASTER:[MONITOR.SRC]MFSUMM.PLI;1 (28)

Page 34

2032 3
2033 3

MC
VC


```
2034 3 IF STAT_IND = YES /* If statistics headings requested, */
2035 3 THEN DO;
2036 4
2037 4 /*
2038 4 /* Put out statistics column headings for stats (TOT AVE MIN MAX).
2039 4 */
2040 4
2041 4 STHEAD_COL = 20 + (19 * COLS_TO_DO) - 1; /* Calculate column number */
2042 4 IF COLS_TO_DO = 0 /* If no data columns, */
2043 4 THEN STHEAD_COL = STHEAD_COL + 9; /* then move stats columns to the right */
2044 4 CALL SCR$SET_CURSOR(7,STHEAD_COL); /* Explicitly set cursor */
2045 4 PUT_LEN = MFS_STHEAD1_STR.L; /* Length of stats head control string */
2046 4 FAOL_REQUESTED = YES; /* Run it through FAOL */
2047 4 OUTP_REQUESTED = NO; /* ... but don't output it yet */
2048 4 CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,MFS_STHEAD1_STR.S.); /* Send 1st stats heading line to SCRPKG */
2049 4 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2050 4
2051 4 CALL SCR$SET_CURSOR(8,STHEAD_COL); /* Explicitly set cursor */
2052 4 PUT_LEN = MFS_STHEAD2_STR.L; /* Length of stats head control string */
2053 4 CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,MFS_STHEAD2_STR.S.); /* Send 2nd stats heading line to SCRPKG */
2054 4 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2055 4
2056 4 END;
2057 3
2058 3 RETURN(NORMAL);
2059 3
2060 3 END MFSUM_HEADINGS;
2061 2
2062 2 END MF_SUMMARY_EVENT;
2063 1
```

```
2064 1 MF_REQ_CLEANUP: Procedure; /* M.F. Summary cleanup procedure */
2065 1
2066 1 /*
2067 1 /*++
2068 1 /*
2069 1 /* FUNCTIONAL DESCRIPTION:
2070 1 /*
2071 1 /* MF_REQ_CLEANUP
2072 1 /*
2073 1 /* Called by MFSUM_REQUEST, either in mainline path, or
2074 1 /* from its condition handler, to close files, free
2075 1 /* virtual memory and generally release acquired resources.
2076 1 /*
2077 1 /* INPUTS:
2078 1 /*
2079 1 /* None
2080 1 /*
2081 1 /* OUTPUTS:
2082 1 /*
2083 1 /* None
2084 1 /*
2085 1 /* ROUTINE VALUE:
2086 1 /*
2087 1 /* None
2088 1 /*
2089 1 /*--
2090 1 /*/
2091 1
2092 1 Declare
2093 1 MFS_FREE_MEM ENTRY RETURNS(FIXED BINARY(31)); /* MONITOR MACRO-32 routine to issue LIB$FREE_VM's */
2094 1
2095 1
2096 1 IF CTRLZ_CHAN ^= 0 /* If CTRL/C and CTRL/Z handlers present, */
2097 1 THEN DO; /* Get rid of them */
2098 1 CALL = SYSS$CANCEL(CTRLZ_CHAN); /* ... and indicate their disestablishment */
2099 1 CTRLZ_CHAN = 0;
2100 1 END;
2101 1
2102 1 IF M->MRB$V_INP_CL_REQ /* If cleanup required for an input file, */
2103 1 THEN CALL = INPUT_CLEANUP(); /* ... then do cleanup for it */
2104 1
2105 1 IF M->MRB$V_SUM_CL_REQ /* If summary cleanup required, */
2106 1 THEN DO; /* Get /SUMMARY file spec ptr to MRB for call */
2107 1 M->MRB$A_SUMMARY = MFS$A_SUMMARY; /* ... and do it */
2108 1 CALL = SUMMARY_CLEANUP();
2109 1 END;
2110 1
2111 1 CALL = MFS_FREE_MEM(); /* Free CDB$A_SUMBUF and CDX$A_SELIDTABLE */
2112 1 FREE MFS; /* Free Multi-File Summary Block */
2113 1
2114 1 RETURN;
2115 1
2116 1 END MF_REQ_CLEANUP;
2117 1
2118 1 END MFSUM_REQUEST;
```


MFSUM REQUEST
V04-000

⁶
16-SEP-1984 02:18:24
5-SEP-1984 15:09:17

VAX-11 PL/I X2.1-273
DISK\$VMSMASTER:[MONITOR.SRC]MFSUMM.PLI;1 (30)

Page 37

COMMAND LINE

PLI/LIS=LISS:MFSUMM/OBJ=OBJ\$:MFSUMM MSRC\$:MFSUMM+LIB\$:MONLIB/LIB

0240 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

